Radiology: Artificial Intelligence

EDITORIAL

Magician's Corner: 2. Optimizing a Simple Image Classifier

Bradley J. Erickson, MD, PhD

From the Department of Radiology, Mayo Clinic, 200 First St SW, Rochester, MN 55905. Received July 1, 2019; revision requested July 19; revision received July 25; accepted August 14. **Address correspondence to** the author (e-mail: *bje@mayo.edu*).

Conflicts of interest are listed at the end of this article.

Radiology: Artificial Intelligence 2019; 1(5):e190113 • https://doi.org/10.1148/ryai.2019190113 • Content code: IN

The goal of this series of articles is to educate and enable those interested in the application of deep learning to medical images. The first article focused on setting up an environment where one can execute a complete deep learning application to classify radiologic images (1). This article examines that application in more detail; we explore some of the parameters that can be adjusted and their effect on the model's performance.

As a reminder of how to get your deep learning environment running, open the Google Chrome web browser to the Colab website (*https://colab.research.google.com*). Select the File > Open Notebook menu option, select the "Github" tab at the top, enter "RSNA" into the search field, and then select the repository "RSNA/Magicians-Corner." Select the entry called "MedNISTClassify.ipynb" to load the notebook. Click the arrow at the left of the first cell to run it (you may need to accept that it is not Google code). Also make sure you have a runtime with a GPU (Runtime > Change Runtime Type; if you change this setting, you need to restart the runtime (Runtime > Restart All Runtimes…).

Cell 2 loads the data: run it. The next cell (cell 3—go ahead and execute it) lists the directories so you can see the six classes of images that we will be teaching our classifier: CTAbd, CTChest, CTHead, MRBreast, MRBrain, and CXR. Remember, the task we will work on is to classify images into one of these six classes.

Classifier Parameters

We are now ready to start learning about the various parameters that are important when performing deep learning. The first line of code we will focus on is the third line of cell 4:

```
data = ImageDataBunch.from_folder(classes_dir, train=".",
  valid_pct=0.2, ds_tfms=get_transforms(), size=64,
  num_workers=4).normalize(imagenet_stats)
```

The purpose of this line of code is to organize our data (hence, the variable called *data* getting the result of this command) into a form that can be used effectively by the training and testing parts of the algorithm. *ImageDataBunch* is the particular data structure that FastAI uses for image data. FastAI (*https://www.fast.ai/*) has other types of databunch for text data and tabular data, but we won't cover those here. The next portion (*from_folder*) indicates how the *ImageDataBunch* should get the data. Other options that you may wish to explore include:

- *from_csv*: a comma-separated values file lists the file and the class;
- *from_df*: a DataFrame is a data object used by a popular data science library called Pandas (*https://pandas.pydata.org/*); and
- *from_lists*: a file contains the list of names and categories.

Further options can be found in the documentation. This provides a wide variety of ways to get data into your artificial intelligence (AI) tool. Full documentation of the *ImageDataBunch* object can be found at *https://docs.fast.ai/vision.data.html#ImageDataBunch.*

Several parameters are common across the various forms of the *ImageDataBunch* expression that are important to know. The first two parameters in our case are specific to the *from_folder* option: they indicate which folder to use and whether the training data are in a separate folder. In our example, the class for each image was determined by its folder name, where the six folders are found in the directory pointed to by the first parameter, which is classes_dir. The second parameter lets us force the use of a separate directory for training data—we will use the same path we used for the class names, so we just use '.' to point to the images.

In the first article, the *valid_pct* parameter was briefly described as determining the fraction of data that should be used for validation. This important parameter deserves more attention. As noted in the previous article, most deep learning tasks perform supervised learning, which means that examples and the answers (classes) are given to the algorithm. If we give all of our data to the algorithm, it will train the best model, but then we would have no independent data to evaluate the model's performance. If a network learns the specific examples well, but does not learn the features of the examples that will allow it to correctly classify new examples, we say it has "overfit" the data (see Table for a list of important machine learning terms). Without data that have not been used by the network for training, we don't have a good way to determine if overfitting has occurred. Therefore, it is common practice to partition the data into a training set and a validation set (FastAI abbreviates the latter to "valid"). A value of 0.2 means that 20% of the data will be used for validation (evaluating performance); the remaining 80% will be used for training. Values between 20% and 40% are common, and depend on the number of examples. Too many validation examples may leave too few examples to train the algorithm well. Having too few validation examples will reduce the accuracy of the performance estimate. The best percentage to use depends on the amount of data and the strength of the "signal" in the data. For each iteration of

A Glossary of Terms Used in Deep Learning

Data augmentation	Creation of additional examples that are derived from real examples, but which are altered in a way that makes them distinct, but realistic. In the case of images, this might include rotation by small angles, flipping, adding small amounts of noise, altering the contrast or brightness by small amounts.
Model	The set of layers or components used to create a machine learning system. This might include a certain number of convolutional layers with a particular kernel size, pooling layers, fully connected layers, and activation function.
Training set	Training is the process by which a number of examples (the training set) are used to update the weights in a model to produce the best performing network. This commonly involves backpropagation, a process in which the magnitude of error is used to estimate the best way to modify the weights.
Testing set	To determine if a network has only learned the training examples, a separate set of "test" examples (aka test set) are used to evaluate a network. One may measure performance on a test set as a system is trained, but the results must not be fed back to assist in training. In that way, one can observe when the training has achieved best performance. In fact, with too much training, the system can start to degrade, so observing the test set performance during training is important.
Validation set	At times, one may also have a separate validation set that is used only after the system is fully trained, to again test the generalization to nontraining examples. Note also that sometimes "validation set" is used for the above definition of "test set," and the "test set" is used as described here for validation, so it is important to read the description in each article.
Overfit	This occurs when a network learns properties specific to the examples and not to the general problem. This can occur when the network has too many parameters compared with the number of training examples. *Generalization* refers to a system that is not overfit, but has learned the important properties of the training set that are present in examples not present in the training set.
Underfit	This is when there is not sufficient complexity of the model to learn the complexity of the problem that is being learned.
Normalize	This is the process by which examples are made to be similar in a way that preserves the important aspects of the data. It can also be used with a more precise meaning, where the values range from 0 to 1.
Standardize	This is the process by which examples are made to be similar in a way that preserves the important aspects of the data. It can also be used with a more precise meaning, where the values have a mean of 0 and unit deviation (that is, examples that are 1 standard deviation above the mean have a value of 1.0).
Activation function	The mathematical transformation whereby multiple inputs, typically from a previous layer, are converted to an output value that is conveyed to the next layer, or used as the output to make a decision. Nonlinearity of the activation function is a key component of learning complex tasks.
Residual block	Often called a *residual network*, it is a component of a network with two or three layers of nodes with normalization and activation functions, but with an additional 'skip layer'. The purpose of the skip layer is that the input to the residual block is compared with the output of the block, and if the block does not do better than the skip (effectively the identity function), then the layers are skipped. This forces the layers to learn, allowing for better performance with fewer layers. This generally allows for faster training and better generalization.

training, the network model will be evaluated on this validation set to see how it is performing. This performance is not fed back to the algorithm, so the model does not gain any information from the validation set. Instead, this information is used to monitor the progress in training and might be used to decide when training will stop. Often there is a validation set, such as used by FastAI, and a third set called *test,* which is held out until the very end. When you read articles describing training of AI tools, you should pay careful attention to the training, validation, and test sets, and how they are handled.

There are several more parameters for the *ImageDataBunch* function that have default values: if you don't specify them, FastAI will assign values that are typically good. Some may be important to override if you know your data are different from the typical classification problem. The ability to specify a separate *train_data* folder can be valuable for many medical imaging tasks because we often have many images from one patient, and all images from a given patient typically should be in either the training or validation group, but not both. If some sections are placed into the training set and others into the validation set, we are likely to have poor generalization because if there is intrapatient similarity (which is quite likely), the network will be "rewarded" for learning that similarity. Separate training and validation folders can ensure that all images from a given patient are in either one folder or the other.

A fundamental challenge in deep learning is that the algorithms can learn very subtle aspects of each image. A common result is that they can learn the specific example images, rather than the concept or general features that make the correct prediction. Imagine teaching students who had no medical knowledge and who could only hear your dictation about a set of images. They would likely start to associate irrelevant components of the image with the diagnoses you were making unless you explain to them what was important: the common features of that diagnosis among several examples.

Data Augmentation

In the case of deep learning, it is difficult to explain what is important, but one can create many versions of one image where the important components are preserved and the irrel-

evant components are altered. This process of data augmentation (2,3) is used in most deep learning projects because it rarely causes problems, and in cases with limited data it can substantially reduce the chance of overfitting. Example ways to perform data augmentation for images include rotating the image by a few degrees or sliding it left, right, up, or down a few pixels. Depending on the specific anatomy, flipping an image horizontally or vertically might create new versions that are useful for training, but in other cases, flipping an image vertically would be nonsense. For example, an upside-down chest radiograph would not be useful; flipping a chest radiograph horizontally (left-right) would likely make it impossible to diagnose situs inversus. Because FastAI is designed primarily for photographic images, the types of transforms that make sense for photographs may not be appropriate for medical images.

Look now at code cell 4 where the *ImageDataBunch* function is called and note the parameter: *ds_tfms = get_transforms ()*. Those familiar with Python programming will recognize that this is a technique where the named parameter (that is, the name matters, not the order of the parameters) called *ds_tfms* receives a function (*get_transforms*) when training set augmentation is performed. This allows us to control how data augmentation is performed. The documentation page (*https://docs.fast.ai/vision.html#Data-augmentation*) describes the default methods and some built-in options that are available, including:

```
get_transforms(do_flip: bool = True, flip_vert: bool = False,
  max_rotate: float = 10.0, max_zoom: float = 1.1,
  max_lighting: float = 0.2, max_warp: float = 0.2, p_affine:
  float = 0.75, p_lighting: float = 0.75, xtra_tfms:
  Union[Collection[fastai.vision.image.Transform], NoneType] =
  None) -> Collection[fastai.vision.image.Transform].
```

Again, for those unfamiliar with Python, the part before the ':' is the name of the parameter, and after the ':' is the type of variable and its default value (that is, the value that is "assumed" if you don't include the named parameter in your call to the function). If we want to allow rotation of the image from −5 to +5 degrees, and disable left-right flipping (note left-right flipping is called "do_flip" while top-bottom flipping is called flip_vert), and did not want to allow warping or lighting (intensity) changes, we would change the function call to this:

```
my_tfms = get_transforms(do_flip=False, max_rotate=5.0,
  max_warp=0.0, max_lighting=0.0, p_lighting=0.0)
```

If you wish to view some of the images created by your augmentation routine, uncomment the three lines at the end of cell 5 and run cell 5 again. You will see many "augmented" examples from the 1 starting head CT image. Note that the augmentation transformations allow rotation, flipping, and contrast changes, and more that won't be covered here. The FastAI documentation page describes and shows these (*https://docs.fast.ai/vision.transform.html*).

Those unfamiliar with data augmentation should try turning off augmentation and running the whole notebook to see the effect on performance. This is done by uncommenting the line in the middle of cell 5, which sets the transforms function to include nothing: tfms = [[],[]]. In most cases of medical imaging, we have very few examples, and using data augmentation to create these variations is extremely useful to promote generalization. However, it is critical that only sensible augmentations be performed. In some cases, altering intensity may be feasible (eg, US or MRI) while for others, the absolute intensity value has specific meaning (eg, CT or parametric images). The next article in this series will describe image preparation in much more detail.

Normalization

The last parameter shown is normalization. Normalization is commonly used to get the images into a similar intensity range—an important thing for photographic images. Understanding normalization is critical because it can introduce spurious associations: for instance, if most of the images of class "cancer" are "bright," the algorithm might learn that a "bright" image means cancer, rather than learning specific textures mean cancer. The default image normalization algorithm in FastAI sets the mean intensity to 0 and the contrast range such that 1 standard deviation is 1.0 (0 mean/unit deviation). If you refer to definitions in the Table, you will see that purists would call this standardization rather than normalization, while others would say that standardization is an example of normalization. This is a very common normalization approach in AI and reflects the fact that images are represented as floating point values, and that there is no absolute intensity scale. This approach is often reasonable for MR, computed radiographic, and US images where there is no meaning to the intensity scale. However, for CT, the scale does have a meaning, and also, changes in values near 0 HU have more importance than at plus or minus 1000 (eg, if brain tissue is 30 units brighter, that might reflect blood, and 30 units darker might mean stroke, but bone tissue 30 units darker or brighter has no meaning). Our knowledge of medical image data properties allows us to do better than uninformed AI experts, but it does require that we override these defaults. Note here that FastAI does allow one to substitute custom functions, including clipping of pixel values to achieve such special functions as "window/level" values.

In our first example, we passed *imagenet_stats*, which are the statistics from the ImageNet (4,5) collection of photographic images (*www.image-net.org/*). ImageNet's statistics provide a starting point but are not optimal for medical images. In our current example, we will pass no parameter rather than *imagenet_stats*, which forces the calculation of statistics for our examples.

Although we are not performing segmentation in this lesson, you can see that near the bottom of the page on *ImageDataBunch* documentation, there are also several options for reading in segmentation maps for images. We will cover segmentation in a future article.

Preparing for Training

Once our data are set up for training, it is wise to visually confirm that things look right, and that is the purpose of cell 5. Running that cell will display a few images from the training

set and their associated label (class). This is a random sample, so you may not see an image from each class.

Now that the data are prepared, we are ready to start the task of training the model. *Model* is the term used for the structure of the machine learning network—the number of layers, the type of layers, number of nodes in a layer, and so forth. The user must define each of these, though again, FastAI has good starting points. One popular model is the ResNet34 model. This gets its name because it uses residual networks (6) in its model, and there are 34 layers. (ResNet18 and ResNet50 are other popular models with 18 and 50 layers, respectively).

We are ready to start training the model. But when do we stop training? This question is a classic problem in computer-based optimization. It might make sense that we want "the best possible" model, but one cannot prove that any particular network is "the best possible." Instead, we must use trial and error to continually improve the network and set criteria for when to stop. We can stop when there is no longer any improvement over a given number of iterations, when a certain amount of time has elapsed, when a certain level of performance has been reached, or whenever any one of these criteria is met.

How do we measure a model's performance? One metric we can (and do) display is the *error_rate*, which we understand well: it is the number of incorrectly classified cases divided by the total number of test cases. However, we may wish to use a different error function such as when a false-negative is more important to avoid than a false-positive. A common metric in the AI literature is the "loss," which is the normalized difference between the output of the network for all examples used for evaluation versus what the ideal output would have been. Therefore, a perfect network would have a loss of 0. This number has no specific meaning; that is, a value of 0.5 for one problem does not mean this network is better than a loss of 0.5 for a totally different problem.

However, it is common to compute the loss for both the training and validation set. If the loss values for the training set are clearly lower than the validation set, it means that you are overfitting: learning the examples and not generalizing well. If training and validation loss values are about the same, then you have a network that has generalized well. This doesn't mean it performs as well as you wish, but it should perform at about this level when shown new examples from the real world from a similar population.

In this article, you have learned more about the many choices that must be made when training a neural network. Popular tools usually have default values that work pretty well, particularly for photographic images. These values may or may not work well for medical images. Experiment with some of the options: try a range of different validation percentages and see the effect on training and validation performance. Try different augmentation methods and see the impact on accuracy—do the changes result in performance changes you would expect? Try ResNet18 and ResNet50 networks and see their impact on performance. In the next article, we will explore the impact of various data preparation schemes.

Disclosures of Conflicts of Interest: B.J.E. disclosed no relevant relationships.

References

1. Erickson BJ. Magician's corner: how to start learning machine learning. Radiol Artif Intell 2019;1(4):e190072.
2. Tustison NJ, Avants BB, Lin Z, et al. Convolutional neural networks with template-based data augmentation for functional lung image quantification. Acad Radiol 2019;26(3):412–423.
3. Gadermayr M, Li K, Müller M, et al. Domain-specific data augmentation for segmenting MR images of fatty infiltrated human thighs with neural networks. J Magn Reson Imaging 2019;49(6):1676–1683.
4. Li FF, Deng J, Li K. ImageNet: Constructing a large-scale image database [abstr]. J Vis 2010;9(8):1037.
5. Krizhevsky A, Sutskever I, Hinton GE. ImageNet Classification with Deep Convolutional Neural Networks. In: Advances in Neural Information Processing Systems 2. NIPS Proceedings. Red Hook, NY: Curran Associates, 2012; 1097–1105.
6. He K, Zhang X, Ren S, Sun J. Deep Residual Learning for Image Recognition. In: Bajcsy R, ed. Proceedings of the IEEE Conference on Computer Visions and Pattern Recognition. Los Alamitos, Calif: Conference Publishing Services, 2016; 770–778.